Conformant Planning via Classical Planners

Khoi Nguyen
Computer Science Department
New Mexico State University
Las Cruces, NM 88001
Email: knguyen@cs.nmsu.edu

Advisor: Dr. Tran Cao Son

Introduction

Conformant planning is the problem of computing a sequence of actions that achieves a goal in presence of incomplete information about the initial state (Smith and Weld 1998). Recent research shows that conformant planning could be very useful in the construction of finite-state controllers (Bonet, Palacios, and Geffner 2009) and in contingent planning (Albore, Palacios, and Geffner 2009). One of the most difficult issues, that directly affects the performance and scalability of conformant planners, is the size of the initial belief state—which is often exponential in the number of object constants of the problem. We observed that in many problems drawn from the recent *International Planning Competitions (IPC)* and from the literature, the initial belief states of many large instances contain more than 2^{10} states, creating challenges to existing conformant planners.

Various techniques have been developed to deal with the potentially huge size of the belief state. Some planners employ a compact representation of belief states—e.g., CFF (Brafman and Hoffmann 2004), POND (Bryce, Kambhampati, and Smith 2006), CNF (To, Son, and Pontelli 2010). Other planners develop simplification techniques that can reduce the size of the initial belief state, sometimes by several orders of magnitude—as in CPA (Tran et al. 2009) and DNF (To, Pontelli, and Son 2009). Most of these planners search for solutions in the belief state space. An alternative approach has been proposed in (Castellini, Giunchiglia, and Tacchella 2001) and (Kurien, Nayak, and Smith 2002), where the conformant planning problem is viewed as a set of *sub-problems*, which are classical planning problems, and solutions are computed using a two-step approach.

CPLAN, developed in (Castellini, Giunchiglia, and Tacchella 2001), starts by computing a solution for a sub-problem, called a *possible plan*, using a SAT-planner. It then checks whether the possible plan is a solution of the original problem. If it is not, the possible plan is discarded, a new possible plan is generated, and the process continues.

FRAG-PLAN, proposed in (Kurien, Nayak, and Smith 2002), follows a slightly different approach in computing plans. It begins with the computation of a possible plan and then attempts to extend it to a valid plan. During the extension phase, FRAG-PLAN assumes that a conformant plan for k initial states has been found, selects fragments of this plan, and uses them as the seed to find a conformant plan for $k+1$ sub-problems where $k = 1, 2, \ldots, n-1$ and n is the size of the initial belief state. The main concern in FRAG-PLAN is the potentially huge number of backtracking steps that need to be performed. We refer to the approach used in FRAG-PLAN as *generate-and-extend*. The experimental evaluation in (Kurien, Nayak, and Smith 2002) shows that both planners work well in some domains, but their coverage is limited, and both planners do not scale up well.

In my thesis, I propose an alternative approach to the generate-and-extend approach of (Kurien, Nayak, and Smith 2002). I refer to the new approach as *generate-and-complete*. The approach is similar to generate-and-extend, as it first generates a possible plan for a sub-problem and then uses it to construct a valid plan by considering other initial states. In the second phase, my approach *repairs* the possible plan, whenever necessary, to create a possible plan for other sub-problems, one-by-one. The possible plan for the last sub-problem will be checked for being a valid conformant plan. If it is not a solution, a new possible plan for the first sub-problem is generated and the completion process restarted. The key difference between my approach and the approach of FRAG-PLAN lies in that my approach does not fragment a possible plan when it is not necessary, and it does not attempt to incrementally compute a conformant plan for a subset of all subproblems in each iteration. Furthermore, my approach employs the `one-of`-combination technique described in (Tran et al. 2009) to reduce the number of possible initial states that need to be considered.

My thesis also aims at investigating the application of different techniques from classical planning, such as landmark and abstraction, to conformant planning. My goal is to identify methods of using landmarks and stratification to transform a conformant planning problem into an "easier" problem, ideally a classical problem, that can be dealt with efficiently by state-of-the-art conformant planners or classical planners. To achieve this goal, I propose the notion of a viable landmark as a belief state and discuss the complexity of the problem of checking whether or not a belief state is a viable landmark. To address the computational challenge of the problem, I propose to approximate viable landmarks and develop an algorithm for computing these approximations.

Conformant Planning Problem

A conformant planning problem P is specified by a tuple $\langle F, O, I, G\rangle$, where F is a set of propositions, O a set of action descriptions, I a set of formulae describing the initial state of the world, and G a formula describing the goal.

A *literal* is a proposition $p \in F$ or its negation $\neg p$. $\bar{\ell}$ denotes the complement of the literal ℓ—i.e., $\bar{\ell} = \neg\ell$, where $\neg\neg p = p$ for $p \in F$. For a set of literals L, $\bar{L} = \{\bar{\ell} \mid \ell \in L\}$, and L is often used to represent $\wedge_{\ell\in L}\ell$.

A set of literals X is *consistent* if there exists no $p \in F$ such that $\{p, \neg p\} \subseteq X$. A *state* s is a consistent and *complete* set of literals, i.e., s is consistent, and for each $p \in F$, either $p \in s$ or $\neg p \in s$. A *belief state* is a set of states. A set of literals X satisfies a literal ℓ (resp. a set of literals Y) iff $\ell \in X$ (resp. $Y \subseteq X$).

Each action a in O is associated with a precondition, denoted by $pre(a)$, and a set of conditional effects of the form $\psi \rightarrow \ell$ (denoted by $a : \psi{\rightarrow}\ell$), where $pre(a)$ and ψ are sets of literals and ℓ is a literal. We often write $a : \psi \rightarrow \ell_1, \ldots, \ell_k$ as a shorthand for the set $\{a : \psi \rightarrow \ell_1, \ldots, a : \psi \rightarrow \ell_k\}$.

The initial state I is a set of literals, `one-of` clauses (each of the form $\texttt{one-of}(\psi_1, \ldots, \psi_n)$), and `or` clauses (each of the form $\texttt{or}(\psi_1, \ldots, \psi_m)$), where each ψ_i is a set of literals.

A set of literals X satisfies the `one-of` clause $\texttt{one-of}(\psi_1, \ldots, \psi_n)$ if there exists some i, $1 \leq i \leq n$, such that $\psi_i \subseteq X$ and for every $j \neq i$, $1 \leq j \leq n$, $\overline{\psi_j} \cap X \neq \emptyset$. X satisfies the `or` clause $\texttt{or}(\psi_1, \ldots, \psi_m)$ if there exists some $1 \leq i \leq m$ such that $\psi_i \subseteq X$.

By $ext(I)$ we denote the set of all states satisfying every literal in I, every `one-of` clause in I, and every `or` clause in I. E.g., if $F{=}\{g, f, h\}$ and $I{=}\{\texttt{or}(g, h), \texttt{one-of}(f, h)\}$ then $ext(I) = \{\{g, h, \neg f\}, \{g, \neg h, f\}, \{\neg g, h, \neg f\}\}$.

The goal G is a collection of literals and `or` clauses.

Given a state s and an action a, a is executable in s if $pre(a) \subseteq s$. A conditional effect $a : \psi \rightarrow \ell$ is applicable in s if $\psi \subseteq s$. The set of effects of a in s, denoted by $e_a(s)$, is defined as: $e_a(s) = \{\ell \mid a : \psi \rightarrow \ell \in O \text{ is applicable in } s\}$. The execution of a in a state s results in a successor state $succ(a, s)$, where $succ(a, s) = (s \cup e_a(s)) \setminus \overline{e_a(s)}$ if a is executable in s, and $succ(a, s) = \mathbf{failed}$, otherwise. Using this function, we define $\widehat{succ}$ for computing the state resulting from the execution of a sequence of actions $\alpha = [a_1, \ldots, a_n]$: $\widehat{succ}(\alpha, s) = s$ if $n = 0$; $\widehat{succ}(\alpha, s) = succ(a_n, \widehat{succ}([a_1, \ldots, a_{n-1}], s))$ if $n > 0$; and $\widehat{succ}(\gamma, \mathbf{failed}) = \mathbf{failed}$ for any sequence of actions γ. For a belief state S and action sequence α, let $\widehat{succ}^*(\alpha, S) = \{\widehat{succ}(\alpha, s) \mid s \in S\}$ if $\widehat{succ}(\alpha, s) \neq \mathbf{failed}$ for every $s \in S$; and $\widehat{succ}^*(\alpha, S) = \mathbf{failed}$, otherwise. α is a *solution* of P iff $\widehat{succ}^*(\alpha, ext(I)) \neq \mathbf{failed}$ and G is satisfied in every state belonging to $\widehat{succ}^*(\alpha, ext(I))$.

GC[Ω]—A Generate-And-Complete Approach

Intuition

In this section, we present the basic idea behind GC[Ω]. First, we introduce the notion of a sub-problem.

Definition 1. *Let $P = \langle F, O, I, G\rangle$ be a conformant planning problem. For each $s \in ext(I)$, the planning problem $P(s) = \langle F, O, s, G\rangle$ is a* sub-problem *of P. A solution of a sub-problem $P(s)$ of P is called a* possible plan *of P.*

It is easy to see that, for every $s \in ext(I)$, the problem $P(s)$ is a classical planning problem. The following observation is an obvious consequence of the definition of a solution of a conformant planning problem.

Observation 1. *Let $P = \langle F, O, I, G\rangle$ be a conformant planning problem and $P(s)$ be a sub-problem of P. Then, every solution of P is also a solution of $P(s)$.*

This property has been used in the development of CPLAN (Castellini, Giunchiglia, and Tacchella 2001) and FRAG-PLAN (Kurien, Nayak, and Smith 2002). In essence, CPLAN uses Algorithm 1.

Algorithm 1 CPLAN(P)

```
1: Input: A planning problem P = ⟨F, O, I, G⟩
2: Output: A solution for P
3: repeat
4:    Compute a possible plan α of P
5:    if α is a solution of P then
6:       return α
7:    end if
8: until every possible plan of P has been considered
9: return failed
```

The authors of FRAG-PLAN observed that CPLAN immediately eliminates a possible plan α of P from consideration when it is not a solution of P, indicating that the action sequence is "useless"—nevertheless, there might be fragments of α that are useful. This led to the development of FRAG-PLAN, which tries to use several possible combinations of fragments of α to construct a solution before dismissing α as useless. While the idea seems reasonable, the approach has some issues of its own. First, the extension phase needs to decide which combinations of fragments should be used. Second, backtracking is required whenever the current combination is deemed not promising. This is problematic, since the number of possible combinations of fragments of an action sequence is exponential in its size (or in the length of the plan).

In my thesis, I propose a compromise between the approaches of FRAG-PLAN and CPLAN—I employ the two phases of FRAG-PLAN but simplify its extension phase by:

- Considering the possible plan as a single fragment when the extension phase starts; and
- Extending the possible plan to generate a *new possible plan* for other initial states.

The first item removes the burden of having to decide which fragments should be used and what is the order among them (reducing backtracking). This also avoids the rigidity of FRAG-PLAN, that imposes a fixed ordering among fragments in the newly generated plan, and thus requires backtracking when the placement of the fragments is not suitable even though the fragments are useful. The second item relaxes the requirement of immediately generating a conformant plan—by focusing instead on possible plans for the various initial states.

Example 1. Consider the problem

$$P = \langle\{f, p, q, r, h\}, O, I, \{h\}\rangle$$

where

$$O = \left\{ \begin{array}{ll} a : \top \rightarrow p, r & b : \top \rightarrow q, \neg f \\ c : \top \rightarrow \neg f, \neg q, r & k : \top \rightarrow h \end{array} \right\}$$

with $pre(a) = \{q\}$, $pre(b) = \{f\}$, $pre(c) = \{f, q\}$, and $pre(k) = \{r\}$; and

$$I = \{\texttt{one-of}(q, \neg q), \neg p, \neg r, f, \neg h\}.$$

Here, $ext(I) = \{s_0, s_1\}$ with $s_0 = \{q, \neg p, \neg r, f, \neg h\}$ and $s_1 = \{\neg q, \neg p, \neg r, f, \neg h\}$ ($\top$ stands for true).

Let us assume that s_0 is selected to start the search for a solution. Let us consider two scenarios:

- The possible plan $\alpha_1 = [a; k]$ is generated. α_1 is not a solution of P because it is not a solution of $P(s_1)$. Thus, we will attempt to find a solution for $P(s_1)$ which has α_1 as a subsequence. This is done by executing α_1 from s_1. Since $pre(a)$ is not satisfied in s_1, we would like to find a plan that achieves $pre(a)$ from s_1. This process yields $[b]$. If we insert b before a, we obtain the sequence $\beta = [b; a; k]$ which is executable in s_1. Incidentally, β is also a solution of $P(s_1)$—i.e., β is a possible plan of $P(s_1)$.

  A validity test reveals that β is indeed a solution of P, and no other possible plans need to be explored.
- Let us assume that $\alpha_2 = [c; k]$ is generated instead. We can easily check that α_2 is not a solution of P, since it is not a solution of $P(s_1)$. Again, we will try to create a solution for $P(s_1)$ which has α_2 as a subsequence. Similarly to the previous scenario, we would like to find a plan that achieves $pre(c)$ from s_1. This will be unsuccessful, since the only action that can generate q, a precondition of c, is action b. However, b will make f false, and there is no action that can generate f. We can quickly dismiss α_2 and request another possible plan of $P(s_0)$. □

Formalizing the Algorithm

The high-level idea of our approach, as discussed above, relies on searching for a conformant plan by inserting actions into a possible plan. The two critical issues are: (*i*) *where* to insert an action or an action sequence; and (*ii*) *how* to determine them. This section will address these issues.

In order to describe the overall algorithm, we need to introduce some additional notation. By $reduct(P)$ we denote the set of initial states obtained by applying the `one-of`-combination technique of (Tran et al. 2009). We note that this technique helps in reducing the size of the initial belief state for several conformant planning problems.

Our algorithm has two parameters—the classical planner used to compute possible plans, denoted by Ω, and the conformant planning problem P. Observe that all state-of-the-art sound and complete classical planners have the following properties: they return (*a*) One or some solutions of the problem if the problem is solvable; (*b*) **Failed** if the problem is unsolvable. As such, we can assume that Ω is a sound and complete classical planner. For the sake of simplicity, we denote with $\Omega(X)$ the set of solutions of the planning problem X returned by Ω; $\Omega(X) = \{\textbf{failed}\}$ if X is unsolvable.

Algorithm 2 GC$[\Omega](P)$

```
1: Input: A planning problem P = ⟨F, O, I, G⟩
2: Output: A solution for P
3: Let Σ = [s_0, ..., s_n] = reduct(P)
4:                          {Compute the set of initial states }
5:                                {that need to be considered}
6: Compute Sol = Ω(P(s_0))
7: if Sol = {failed} then
8:    return failed
9: end if
10: while Sol ≠ ∅ do
11:    Select α_{s_0} ∈ Sol          {Obtain a solution of P(s_0)}
12:    if α_{s_0} is a solution of P then
13:       return α_{s_0}
14:    else
15:       β = completion(α_{s_0}, P, Σ, 1)
16:       if β is a solution of P then
17:          return β
18:       end if
19:    end if
20:    Sol = Sol \ {α_{s_0}}
21: end while
22: return unknown
```

At first sight, Algorithm 2 is fairly similar to Algorithm 1. However, they differ in three key aspects. *First,* Algorithm 2 considers only a subset of all possible initial states whenever possible (Line 3), i.e., when the `one-of`-combination is applicable for problem P. *Second,* it attempts to construct a possible plan for other initial states in the reduced set of initial states (Line 15). *Third,* it repeatedly requests for a possible plan from the same initial states. Algorithm 2 differs from the algorithm in FRAG-PLAN in its key step, Line 15, where a new possible plan is constructed.

Intuitively, the algorithm explores the solution space of sub-problem $P(s_0)$ of P; each plan α_{s_0} of $P(s_0)$ is considered (Line 11) and an attempt is made to "repair" it, so that it becomes a plan for other subproblems $P(s_i)$, $s_i \in reduct(P)$. It returns **failed** if Ω indicates that $P(s_0)$ is not solvable. The procedure $completion(\alpha, P, \Sigma, Index)$, executed in Line 15, is the actual algorithm that encodes the process of completing the action sequence α into a potential solution of P, as described in Examples 2-**??**. If the completion fails, another solution for $P(s_0)$ is considered and the process repeated. The algorithm returns **unknown** if it cannot generate a solution.

The procedure $completion(\alpha, P, \Sigma, Index)$ is described in Algorithm 3. Its parameters are the conformant planning problem P, whose initial belief state, represented as a list, is $\Sigma = [s_0, \ldots, s_n]$, a solution α of $P(s_0)$, and an index used to guide the start of the completion process. The procedure attempts to create solutions α_{s_i} for $P(s_i)$, $i = 1, \ldots, n$.

For each iteration of the **for-loop** in Lines 5–27, the algorithm constructs a solution of the sub-problem $P(s_i)$ from the solution $\alpha_{s_{i-1}}$ of $P(s_{i-1})$, by inserting actions into $\alpha_{s_{i-1}}$. To achieve this, the algorithm starts with the state s_i (Line 7) and an empty plan and considers each action a in $\alpha_{s_{i-1}}$ (Lines 9–21):

- **Task 1**: inserts a sequence of actions before a (loop 9–

Algorithm 3 $completion(\alpha, P, \Sigma, Index)$

1: **Input**: α–a solution of $P(s_0)$
$P = \langle F, O, I, G\rangle$–conformant planning problem
$\Sigma = [s_0, \ldots, s_n]$–list of initial states of P
$Index$–the index for starting the completion
2: **Output**: A possible solution for P
3: Let $\alpha_{s_{Index-1}} = \alpha$
4: Initialize $\alpha_{s_i} = []$ for $i = Index, \ldots, n$
5: **for** $i = Index$ to n **do**
6: {completion of α for the states $s_{Index}, \ldots, s_n$}
7: $s = s_i$ {current state}
8: Assume that $\alpha_{s_{i-1}} = [a_0, \ldots, a_{last}]$
9: **for** $j = 0$ to $j = last$ **do**
10: $tGoal = pre(a_j)$ {create a temporary goal}
11: **if** $tGoal = \emptyset$ **then**
12: $E = \{a_j : \psi \rightarrow l \mid a_j : \psi \rightarrow l \in O$ is applicable in $\widehat{succ}([a_0, \ldots, a_{j-1}], s_{i-1})\}$
13: $tGoal = tGoal \cup (\bigcup_{[a_j:\psi\rightarrow l]\in E} \psi)$
14: **end if**
15: $\gamma \in \Omega(\langle F, O, s, tGoal\rangle)$
16: **if** $\gamma =$ **failed then**
17: **return failed**
18: **end if**
19: $\alpha_{s_i} = \alpha_{s_i} \circ \gamma \circ [a_j]$ {update current plan}
20: $s = succ(a_j, \widehat{succ}(\gamma, s))$ {update current state}
21: **end for**
22: $\delta \in \Omega(\langle F, O, s, G\rangle)$
23: **if** $\delta =$ **failed then**
24: **return failed**
25: **end if**
26: $\alpha_{s_i} = \alpha_{s_i} \circ \delta$ {update current plan}
27: **end for**
28: **return** α_{s_n}

21), so that *(1)* a is executable and *(2)* the execution of a maintains the effects of a if $pre(a) = \emptyset$, i.e., a is always executable. To achieve this, the algorithm creates a temporary goal ($tGoal$) and modifies it accordingly (line 10) and (lines 11–14).
There are two reasons behind this task. First, most classical planners are fairly sophisticated and do not generate redundant actions. As such, the existence of a in $\alpha_{s_{i-1}}$ is (almost always) necessary for achieving the goal in s_{i-1}; thus, its effects need to be maintained for the final plan to be a conformant plan. Second, planning from the current state s to achieve $tGoal$ is expected to be significantly simpler compared to planning to achieve the final goal from s.

- **Task 2**: makes sure that the final sequence of actions α_{s_i} achieves the goal of $P(s_i)$ (Line 22)—and this may require adding extra actions at the end of α_{s_i}.

Observe that each α_{s_i} is a solution of $P(s_i)$ but it is possible that none of the α_{s_i} is a solution of P. This due to the fact that the insertion of actions into $\alpha_{s_{i-1}}$ does not guarantee that α_{s_i} remains a solution of $P(s_{i-1})$. This is also the reason why Algorithm 2 includes the test in line 16.

On Improving Conformant Planners by Analyzing Domain-Structures

Viable Landmarks in Conformant Planning

In this section, we define the notion of a viable landmark for conformant planning problems. By definition, the search for a conformant plan is done in the space of belief states. A straightforward generalization of a landmark to conformant planning would be a literal which is true in one of the belief states generated during the execution of any solution from the initial belief state. This is quite restrictive though.

Example 2. *Consider* $P = \langle\{f, g, h\}, O, I, \{h\}\rangle$ *where* $I = \{\texttt{one-of}(f, g), \texttt{one-of}(h, \neg h)\}$ *and* O *contains* $a : f, \neg g \rightarrow \neg f, \neg g$*;* $a : \neg f, g \rightarrow f, g$*;* $b : \neg f, \neg g \rightarrow h$*;* $b : f, g \rightarrow h$*;* $c : f \rightarrow \neg f$*; and* $c : g \rightarrow \neg g$*. Furthermore,* $pre(a) = pre(b) = pre(c) = \top$.

It is easy to see that $[a, b]$ *is a solution of* P*. Yet, there exists no literal* ℓ *different from* h *that is true during the execution of* $[a, b]$*. In other words, this problem has only a trivial landmark,* h*, according to the proposed generalization.*

It is also easy to see that $[c, b]$ is another solution of P in Example 2 and for every state $s \in succ^*(c, ext(I))$, $\neg f \in s$, and $\neg g \in s$. In other words, the uncertainty with regards to the clause $\texttt{one-of}(f, g)$ has been removed from $succ^*(c, ext(I))$. Moreover, $succ^*(c, ext(I))$ has fewer elements than $succ^*(a, ext(I))$. This means that for conformant planners employing the cardinality heuristic, the solution $[c, b]$ is likely to be returned as the first solution of P. This motivates us to define the notion of a viable landmark for a conformant problem as a belief state which (*i*) can easily be reached or predicted; and (*ii*) from which there is a sequence of actions that reaches the goal. In the above example, $succ^*(c, ext(I))$ would be a viable landmark, which can be reached by the sequence $[c]$ and from which there is the sequence $[b]$ that reaches the goal. Let us formalize these ideas.

Definition 2. *Let* S *be a belief state,* Ω *be a belief state, and* o *be an* `one-of` *clause. We say that* S *is a* convergent point *of* Ω *for* o*, denoted by* $\Omega \rightsquigarrow S[o]$*, if for every literal* ℓ *appearing in* o*,* $S \models \ell$ *or* $S \models \overline{\ell}$*, and for every state* $s \in \Omega$*, there exists an action sequence* α_s *such that* $\widehat{succ}(\alpha_s, s) \in S$.

In Example 2, $succ^*(c, ext(I))$ is a convergent point of $ext(I)$ for $\texttt{one-of}(f, g)$. A convergent point is only useful for the search of a solution if there exists a action sequence α that leads to this point from all possible initial states. We therefore define the notion of an abstract point as follows.

Definition 3. *Let* S *be a belief state,* Ω *be a belief state,* α *be an action sequence, and* o *be an* `one-of` *clause. We say that* S *is an* abstract point *of* Ω *for* o *w.r.t.* α*, denoted by* $\Omega \overset{\alpha}{\rightsquigarrow} S[o]$*, if* $\Omega \rightsquigarrow S[o]$ *and* $\widehat{succ}^*(\alpha, \Omega) \subseteq S$.

We call α *an* abstract path for o from Ω to S. S *is an* abstract point *of* Ω *for* o *if there exists an action sequence* α *such that* $\Omega \overset{\alpha}{\rightsquigarrow} S[o]$.

We have that $ext(I) \overset{[c]}{\rightsquigarrow} succ^*(c, ext(I))[\texttt{one-of}(f, g)]$ for Example 2. The following proposition holds.

Proposition 1. *Let S be a belief state, Ω be a belief state, α be an action sequence, and o be an* `one-of` *clause. Then, $\Omega \overset{\alpha}{\rightsquigarrow} S[o]$ implies $\Omega \overset{\alpha}{\rightsquigarrow} \widehat{succ}^*(\alpha, \Omega)[o]$.*

We will often say that α is an abstract path for a planning problem $P = \langle F, O, I, G\rangle$ if there exists an `one-of`-clause o in I such that $ext(I) \overset{\alpha}{\rightsquigarrow} \widehat{succ}^*(\alpha, ext(I))[o]$.

Definition 4. *Let $P = \langle F, O, I, G\rangle$ be a conformant planning problem. A belief state S is called a* viable landmark *of P if S is an abstract point of $ext(I)$ for some* `one-of` *clause in I and the problem $P' = \langle F, O, S, G\rangle$ has a solution.*

It is easy to see that we achieve our stated objective of having $succ^*(c, ext(I))$ as a viable landmark of P for Example 2. The next proposition follows immediately from the definition of a viable landmark.

Proposition 2. *Let $P = \langle F, O, I, G\rangle$ be a conformant planning problem and S be a viable landmark of P. Then, there exists an action sequence α such that for every solution β of the problem $P' = \langle F, O, S, G\rangle$, $\alpha \circ \beta$ is a solution of P.*

β in Prop. 2 is referred to as a *goal path* w.r.t. S. Prop. 2 indicates that we could solve a conformant planning problem $P = \langle F, O, I, G\rangle$ by (*i*) finding a viable landmark S of P; and (*ii*) solving the problem $P' = \langle F, O, S, G\rangle$. Prop. 1 implies that instead of S, we can find an action sequence α and solve the problem $P'' = \langle F, O, \widehat{succ}^*(\alpha, ext(I)), G\rangle$. By definition of a viable landmark, we know that there exists an `one-of` clauses o in I such that every literal ℓ in o is known to be either true or false in S (or in $\widehat{succ}^*(\alpha, ext(I))$). It means that the uncertainty in o has been removed. Observe that, in general, there is no guarantee that the search for a solution from the new belief state S (or $\widehat{succ}^*(\alpha, ext(I))$) will be easier than the search for a solution from the initial belief state $ext(I)$. However, if α only changes the value of literals in o, then P'' is *closer* to a classical planning problem than the original problem P, and thus it could be easier than P, due to the fact that classical planning has a lower complexity than conformant planning (Baral, Kreinovich, and Trejo 2000).

A main obstacle in applying the above idea lies in the fact that finding a viable landmark of P is also not a simple task.

Let *RLandmark* be the problem: given a conformant planning problem $P = \langle F, O, I, G\rangle$ and a belief state S, determine whether S is a viable landmark of P. Because determining whether S is a viable landmark requires checking for a plan from S to G, the complexity of *RLandmark* is at least as hard as the conformant planning problem (i.e., Σ_2^P (Baral, Kreinovich, and Trejo 2000)).

The above complexity result shows that attempting to find an arbitrary viable landmark and using it in the search for a solution is likely not a good idea. However, Prop. 1 implies that a viable landmark is associated with an abstract point and an action sequence (an abstract path). Thus, we can approximate a viable landmark by an abstract point.

Approximating Abstract Points

We observe that for a belief state of Ω for an `one-of` clause, there are two cases where we can predict the viable landmark easily:

- The viable landmark o is one of the states in Ω.
- The viable landmark o lies outside Ω and there is a common path to reach o from every state $s \in \Omega$.

Let us generalizes these two cases.

Definition 5. *Let $\Omega = \{s_1, \ldots, s_n\}$ be a belief state and o an* `one-of` *clause. We say that o is* closed-convergent *in Ω if there exists an $s_i \in \Omega$ and an action sequences α such that $\widehat{succ}(\alpha, s_j) = s_i$ for every j such that $1 \leq j \leq n$. We say that o is* open-convergent *in Ω if there exist a belief state S and an action sequence α such that $\widehat{succ}(\alpha, s_i) \in S$ for every j such that $1 \leq j \leq n$.*

In the rest of this section, we will develop a greedy algorithm for identifying possible abstract points of a planning problem. Before we discuss the idea in more details, let us introduce the following notation. For an `one-of` clause $o = (l_1, \ldots, l_n)$, by an interpretation of o we denote a set of literals δ such that (*i*) for every literal $l \in \delta$, $l \in lit(o) \cup \overline{lit(o)}$; and (*ii*) for each i, $\{l_i, \overline{l_i}\} \cap \delta \neq \emptyset$ and $\{l_i, \overline{l_i}\} \setminus \delta \neq \emptyset$. For a state s, let $s|_o = s \cap (lit(o) \cup \overline{lit(o)})$. For a belief state Ω, let $\Omega|_o = \{s|_o \mid s \in \Omega\}$. It is easy to see that the following holds.

Proposition 3. *Let Ω be a belief state, o be an* `one-of` *clause, and δ be an interpretation of o. If α is a plan achieving δ from Ω, then $\Omega \overset{\alpha}{\rightsquigarrow} \widehat{succ}^*(\alpha, \Omega)[o]$.*

Although simple, the above proposition shows that an abstract point can be characterized by an interpretation of the `one-of` clause in consideration. This is equivalent to say that if the intention is to reduce the uncertainty caused by an `one-of` clause o in the belief state Ω, then it is reasonable to focus on the set of interpretations of o that could be reached from Ω, i.e., on $\Omega|_o$ and the interpretations reachable from $\Omega|_o$. Thus, we can reduce the original domain to a domain related to o and analyze this domain to look for an abstract point.

Given a planning domain (F, O), the reduced domain of (F, O) by the abstraction of o, denoted by $Ab(F, O, o)$, is the planning domain (F', O') obtained from (F, O) where

- $a{:}\phi|_o \rightarrow \psi|_o \in O'$ iff $a{:}\phi \rightarrow \psi \in O$ and $pre(a) \cup \phi$ satisfies o;
- for each a in O', $pre(a)$ is changed to $pre(a)|_o$;
- F' contains l iff $l \in F$ and l or $\neg l$ occurs in some conditional effect in O'.

Let $Ab(F, O, o)$ be the reduced domain of (F, O) by the abstraction of o. The transition graph of $Ab(F, O, o)$, denoted by $G(o)$, is defined as a labeled graph (V, E) where each $\delta \in V$ is an interpretations of o and $(\delta, a, \delta') \in E$ iff there exits a in $Ab(F, O, o)$ such that $succ(a, \delta)$ satisfies δ'. The reduced graph of a belief state Ω, denoted by $G(\Omega, o)$, is obtained by removing from $G(o)$ every node δ that is not reachable from a node in $\Omega|_o$ and the edges coming in or out from these nodes. Fig. 1 shows the transition graph of $Ab(F, O, o)$ for different simple problems. The label in a node specifies what is true in the interpretation. Labeled links between nodes represent actions. The dotted oval contains the interpretations satisfying the `one-of` clause.

Observe that Defs 5 implies that an interpretation δ of o that is reachable by every other interpretations in $\Omega|_o$ is a

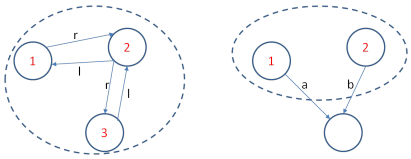

Figure 1: Transition Graph for Reduced Domain

candidate for being an abstract point. So, computing a candidate convergent point of $\Omega|_o$ can be done by (*i*) creating $G(\Omega, o)$; and (*ii*) searching for a node that is reachable from every interpretation satisfying o. As seen in Figure 1 (left), there could be several candidates. Furthermore, checking reachability between the nodes is, theoretically, a computationally expensive task since the $G(\Omega, o)$ might have a number of nodes exponential in the size of the `one-of` clause.

To this end, we develop a greedy method for predicting of an abstract point as follows. For an `one-of` clause o and a belief state Ω, we say that o is *probably closed-convergent* in Ω if for every action a in $Ab(F, O, o)$, a maintains o, i.e.., for every interpretation δ satisfies o, $succ(a, \delta)$ satisfies o. Otherwise, o is *probably open-convergent* in Ω. We then identify an interpretation δ in $G(\Omega, o)$ as follows.

- o *is probably closed-convergent*: selects δ which satisfies o and has the minimal number of outgoing links; and
- o *is probably open-convergent*: selects δ which does not satisfy o and has the maximal number of incoming links.

Result and Future Work

Both approaches, described in previous section, have been applied in conformant planning and results in different planners that are orders of magnitude faster than state-of-the-art conformant planners. The experimental evaluation can be found in (Nguyen et al. 2011) and (Nguyen et al. 2012). These results show that my planners can outperform other planners in scalability and performance. This encourages me to focus my research into two directions: identifying the real-world problems that can be solved by conformant planners; and investigate on optimal conformant planning.

The first objective comes from the observation that my conformant planners have provided significant better performance and scalability when compared to other state-of-the-art planners. Thus, applying my approaches to real-world problems is an immediate consequence. I have experimented with the set of Finite-State Controller problems (Bonet, Palacios, and Geffner 2009). However, these problems were created specifically to be solved using the translation approach in `t0` (Palacios and Geffner 2009). I would like to investigate the application of the *generate-and-complete* on the original Finite-State Controller problem. Similarly, as (Albore, Palacios, and Geffner 2009) has shown that contingent planning problem can be solved through conformant planning, I would like to extend my results to contingent planning.

As for optimal conformant planning, there has been little effort in solving the problem optimally: to the extend of my knowledge, the best optimal planner can only solve only 4 instances of the square center domain(Palacios, Bonet, Darwiche and Geffner 2005) while my planners can solve all 31 instances non-optimally. The main reason for this result is that conformant planners were struggling over performance and scalability even under satisfycing plan condition. As my approaches have overcomed these issues, the next step is to extend these methods to optimal conformant planning. It is worth to note that these approaches employs different greedy strategies which trade completeness for performance. Thus porting these techniques directly to optimal conformant planning is not a good idea in general. In the short future, I would like to identify the source of incompleteness and develop new technique that maintain both completeness and performance.

References

Albore et al. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.

Bonet et al. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In ICAPS.

Baral, C.; Kreinovich, V.; and Trejo, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *AIJ* 122:241–267.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *ICAPS*.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006. Planning Graph Heuristics for Belief Space Search. *JAIR* 26:35–99.

Knoblock, C. 1994. Automatically generating abstractions for planning. *AIJ* 68:43–302.

Kurien, J.; Nayak, P. P.; and Smith, D. E. 2002. Fragment-based conformant planning. In *AIPS*, 153–162.

Nguyen, H.-K.; Tran, D.-V.; Son, T. C.; and Pontelli, E. 2012. On Computing Conformant Plans Using Classical Planners: A Generate-And-Complete Approach. In *ICAPS*.

Nguyen, H.-K.; Tran, D.-V.; Son, T. C.; and Pontelli, E. 2011. On improving conformant planners by analyzing domain-structures. In AAAI.

Héctor Palacios, Blai Bonet, Adnan Darwiche and Hector Geffner 2005. Pruning Conformant Plans by Counting Models on Compiled d-DNNF Representations. In *ICAPS*.

Palacios, H., and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *JAIR* 35:623–675.

Smith, D., and Weld, D. 1998. Conformant Graphplan. In *AAAI*, 889–896.

To et al. 2009. A conformant planner with explicit disjunctive representation of belief states. In *ICAPS*.

To, S. T.; Son, T. C.; and Pontelli, E. 2010. A New Approach to Conformant Planning using CNF. In ICAPS.

Tran et al. 2009. Improving performance of conformant planners: Static analysis of declarative planning domain specifications. In *PADL*, 5418 of *LNCS*, 239–253. Springer.

Castellini, C.; Giunchiglia, E.; and Tacchella, A. 2001. Improvements to sat-based conformant planning. In *Proc. of 6th European Conference on Planning (ECP-01)*.